edX HarvardX - Data Science: Apex Legends Report

Jason Katsaros

2023-10-12

Contents

1 Introduction

Apex Legends is a video game that is very close to my heart. Recently, my brother had the amazing opportunity to pursue a new job in Charlotte, North Carolina, far away from where I currently live. Apex Legends is one way that I get to spend time with my brother, usually on a weekly basis. I always look forward to the time I get to start up a match and just chat with my brother (the game is fun, too). This is why I chose to investigate an aspect of Apex Legends that can possibly be predicted by machine learning. While I know this will not make me better at the game, I look forward to learning and building something around a special pasttime. Through this capstone, I will explain what exactly Apex Legends is, find match data for Apex Legend's main method of gameplay and tidy said data, and create a model to predict the final state of Apex Legends games.

1.1 Motivation

Recently, the company I currently work for put me in an interesting (albeit awkward) position by announcing that I am out of a job by the end of this year. Not one to mope and stand idly by, I took it upon myself to teach myself something new and prepare myself for the next adventure just over the horizon. I find life at its most exciting when I am in the midst of a challenge where I can push through and come out better at the other end. Thus, I decided to participate in the HarvardX: PH125.9x - Data Science program to hone new skills, including but not limited to the R programming language, data science and analysis, and machine learning.

1.2 Overview

Apex Legends("Apex Legends - About" 2019) is a competitive, online first person shooter (FPS) video game developed by Respawn Entertainment and published by Electronic Arts. In Apex Legends, you select a character from a growing cast, each with its own unique passive ability, tactical ability (an ability you can use frequently), and an ultimate ability (a powerful ability you can use infrequently) to beat the other teams (called squads) at some objective, using the vast arsenal of weapons and tools provided to you during each match. While I am not very good at the game yet, I do like to play the character Valkyrie("Apex Legends - Characters" 2019).

While there are many modes of gameplay in Apex (in fact, Respawn Entertainment likes to experiment with different rules, maps, mechanics, etc.), the original mode and focus of this capstone project is the battle royale mode("Apex Legends - Battle Royale" 2019). In battle royale, your squad of three players work together on one big map to eliminate the other nineteen squads (for 60 players total) before they cut you and your teammates out of the game. You can eliminate another player by dealing damage equal to or exceeding their character's health pool plus whatever armor their character might have found and put on during the match. Over the course of the game, a ring closes in on the map, which will deal damage to any player outside of its boundaries, with the purpose of forcing squads closer and closer together, until the ring encompasses everything, putting a final timer on the match. Each match of battle royale has five distinct rings, with the match starting with no ring and ending with the ring covering everything. The five rounds of the ring transition between states at a set period of time within each match and deal different amounts of damage to players to put a timebox on the game. You can see the ring information as shown below("Apex Legends - Endzone & Ring Prediction" 2022):

Round	Time to wait	Time to close	Damage per tick	Ring diameter after closing
1	3 minutes	3 minutes 45 seconds	2	1000 meters
2	2 minutes 45 seconds	45 seconds	3	650 meters
3	2 minutes 15 seconds	45 seconds	10	400 meters
4	1 minute 45 seconds	40 seconds	20	200 meters
5	1 minute 30 seconds	40 seconds	20	100 meters
6	1 minute	2 minutes	25	0.05 meters

In this capstone project, I would like to see if I can use machine learning through R and the `CAReT`(Kuhn and Max 2008), or Classification and Regression Training, package to predict where the final ring will end on a map. While there are also several different maps that put varying points of interest (POI) and obstacles in your path to achieving victory("Apex Legends - Battle Royale - Maps" 2019), this capstone project will only focus on two: World's Edge and Storm Point.

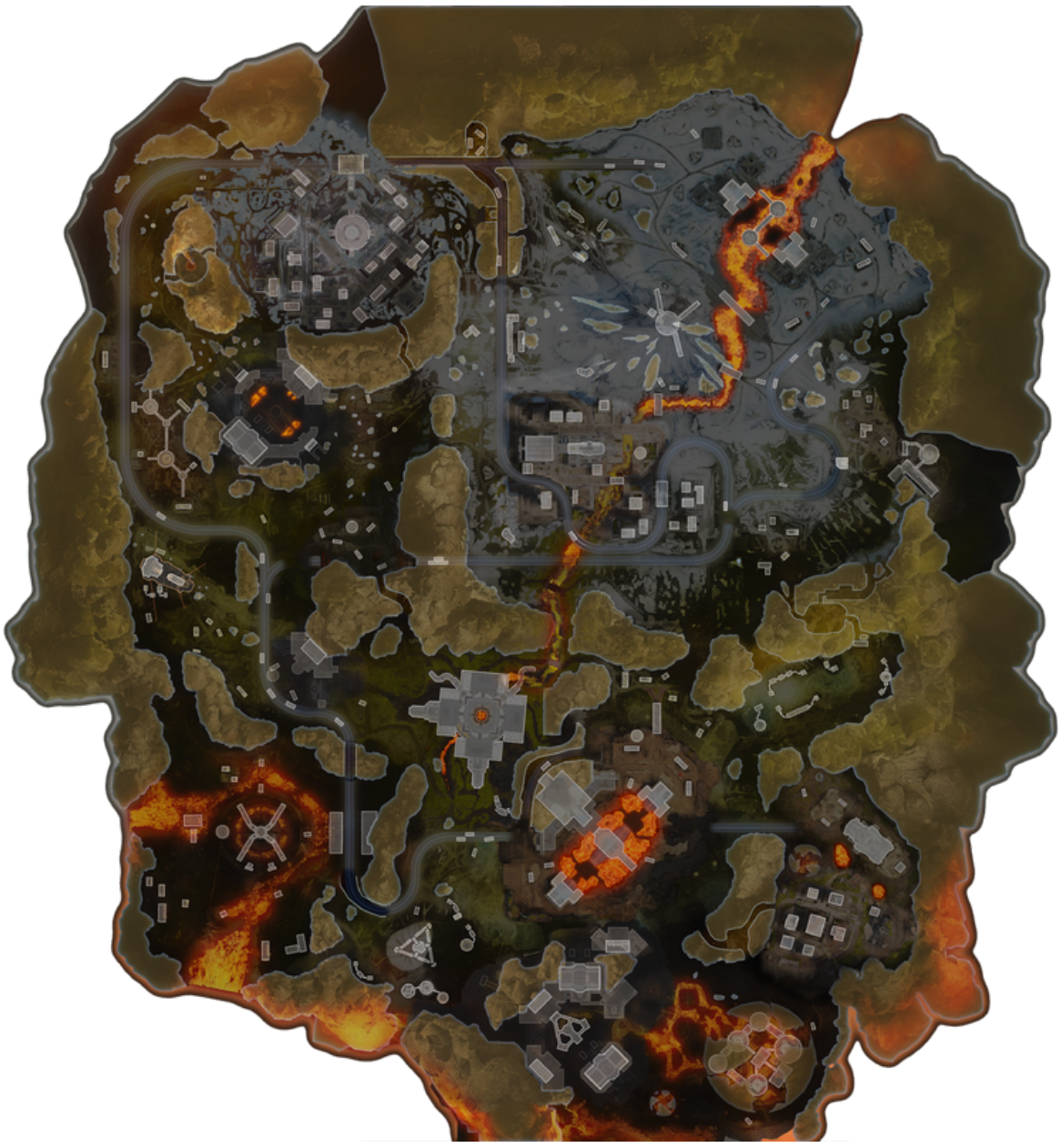


Figure 1: World's Edge Map

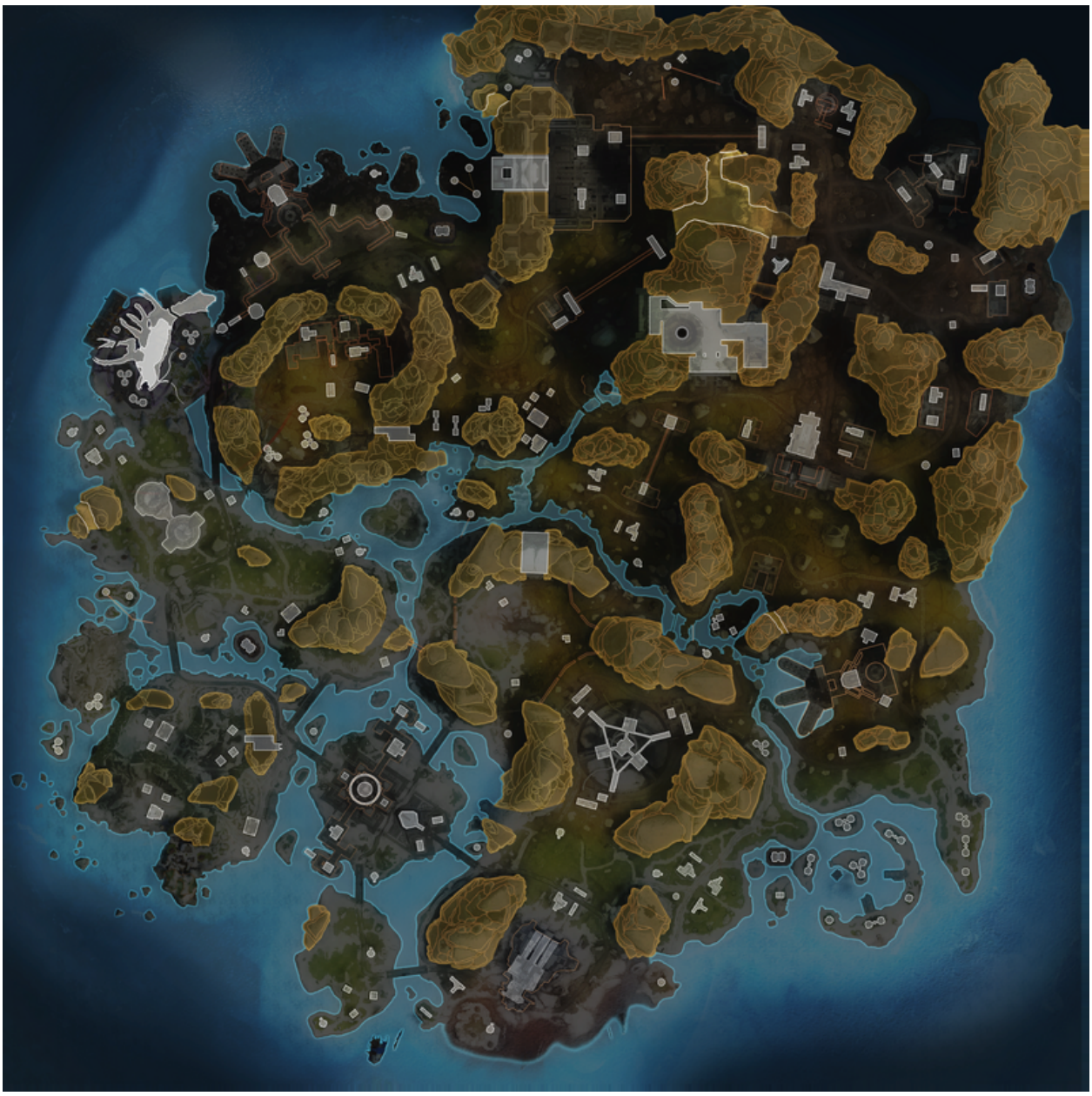


Figure 2: Storm Point Map

2 Data and Analysis

2.1 Set Up

Throughout this capstone project I will use a number of R packages, including the previously mentioned `caret`(Kuhn and Max 2008) package. I will use the `jsonlite`(Ooms 2014) and the `tidyverse`(Wickham et al. 2019) packages for data manipulation, and the `ggplot2`(Wickham 2016), `ggforce`(Pedersen 2022), `ggimage`(Yu 2023), `gganimate`(Pedersen and Robinson 2022), and `gridExtra`(Auguie 2017) packages for visualizing data.

2.2 Initial Data

To start off this capstone project, I was not given any sort of initial data this time around. Instead, I had to go on the hunt for enough Apex Legends match data that I could create a model using said data. I found a GitHub repository, owned and maintained by the user `bluelightgit`("Apex Zone Predict Machine Learning" 2023) that had Apex Legends match data that fit the bill. The match data in question was stored in JSON (JavaScript Object Notation) format. This is slightly different than the CSV (Comma Separated Values) format that we worked with during the HarvardX Data Science course, but I am used to working with JSON data, being a software developer. The `jsonlite`(Ooms 2014) R package makes it very easy to import JSON data. Using the R code shown below, I download the JSON data and store it in my data directory, import the JSON data into R, then transform the JSON data into a tibble.

```
# Download the json file to the "data" directory
json_file <- "data/apex.json"
if(!file.exists(json_file))
  download.file(
    paste0(
      "https://raw.githubusercontent.com/",
      "bluelightgit/apex-zone-predict-machine-learning/",
      "main/zone_datas/zones_data.json"
    ),
    json_file
  )

# Read and parse the json data
json_data <- fromJSON(json_file)

# Transform the json data into a tibble
apex_data <- as_tibble(json_data)

# Save the apex data to an RData file
save(apex_data, file = "rda/apex_data.rda")

head(apex_data)
```

```
## # A tibble: 6 x 5
##   center$x    $y radius stage gameID                           map
##      <int> <int>  <int> <int> <chr>                            <chr>
## 1     6601 11278  13290     0 09914f6562a97a9dc531f9b141fe482e we
## 2     6601 11278   4369     1 09914f6562a97a9dc531f9b141fe482e we
## 3     4956 12420   2367     2 09914f6562a97a9dc531f9b141fe482e we
## 4     5205 13228   1456     3 09914f6562a97a9dc531f9b141fe482e we
## 5     4920 12846    728     4 09914f6562a97a9dc531f9b141fe482e we
## 6     4751 12686    364     5 09914f6562a97a9dc531f9b141fe482e we
```

2.3 Pre-Processing

Before I continue to analyzing the data, I am first going to perform some pre-processing on the Apex Legends match data. Throughout the course of this capstone project, I am going to work with the match data represented in two different ways:

1. A long format, which is easier to use when visualizing data, and
2. A wide format, which is easier to use when training and evaluating the model I am going to create.

2.3.1 Long Data

Firstly, I want to tidy up the Apex Legends match data, which will make viewing the data but also make creating the wide data much easier in the long run. In the long representation of the Apex Legends match data, each row will correspond to a stage of the ring, meaning each game will have five rows.

1. Currently, the x and y coordinates of each stage of the ring within each game in the data set are nested within their own data frame. To make the x and y coordinates easier to access, I want to bring them out from their own nested data frame and into each row.
2. The `map` column is not very human-readable at the moment, with `we` corresponding to `World's Edge` and `sp` corresponding to `Storm Point`. I want to convert the `map` column into a factor and also change the name of said factors to be very clear what map each row matches with.
3. Speaking of factors, I will also make the `stage` and `gameID` columns factors, where `stage` represents the current phase of the ring in an Apex Legends map, and `gameID` represents a unique identifier for each game in the Apex Legends data

```
# Copy the apex data to a new variable for manipulation
apex_data_tidy <- apex_data
# Bring the x and y coordinates of the ring out from a nested data frame
# and into their own columns in the data set
apex_data_tidy <- apex_data_tidy %>%
  mutate(x = center$x, y = center$y) %>%
  select(-center)
# Rename "we" to "World's Edge" and "sp" to "Storm Point" respectively
# Make the "map" column values factors
apex_data_tidy <- apex_data_tidy %>%
  mutate(map = as.factor(ifelse(map == "we", "World's Edge", "Storm Point")))
# Make the "stage" and "gameID" column values factors
apex_data_tidy <- apex_data_tidy %>%
  mutate(stage = as.factor(stage), gameID = as.factor(gameID))

# Save the tidy apex data to an RData file
save(apex_data_tidy, file = "rda/apex_data_tidy.rda")
```

After cleaning up the Apex Legends match data a little bit, there is also some information that I would like to calculate and represent in the data, which I believe will be helpful later on when creating the model for predicting the final ring state in a match.

1. The distance between each ring stage, calculated using the equation to find the length of the hypotenuse of a triangle:

$$\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

where i represents the ring stages, 1 through 5 (0 being the ring stage where there is no ring on the map).

2. The angle between each ring stage, calculated like so:

$$\arctan m_i$$

where m is the slope of the line drawn between the ring stages and i represents the ring stages, 1 through 5 (0 being the ring stage where there is no ring on the map).

```
# Copy the tidy apex data to a new variable for further manipulation
apex_data_long <- apex_data_tidy

# Create and populate a new variable with 0
distance_between <- rep(0.0, nrow(apex_data_long))

# For each ring (and the ring to follow),
# if the current ring is not either
# 1. the ring that does not display (ring 0)
# 2. or the last ring of the match (ring 5)
# then calculate the distance between the two ring stages
for (current_ring in 1:nrow(apex_data_long)) {
  if (
    apex_data_long[current_ring,]$stage != 0 |
    apex_data_long[current_ring,]$stage != 5
  ) {
    distance_between[current_ring] <-
      sqrt(
        (
          apex_data_long[current_ring + 1,]$x -
            apex_data_long[current_ring,]$x
        )^2 +
          (
            apex_data_long[current_ring + 1,]$y -
              apex_data_long[current_ring,]$y
          )^2
      )
  }
}

# Transform any NaN into 0
distance_between[is.nan(distance_between)] <- 0

# Add the new information to the apex_data_long data set
apex_data_long <- apex_data_long %>%
  mutate(distance_between = distance_between)

# Create and populate a new variable with 0
angle_between <- rep(0.0, nrow(apex_data_long))

# For each ring (and the ring to follow),
# if the current ring is not either
# 1. the ring that does not display (ring 0)
# 2. or the last ring of the match (ring 5)
# then calculate the angle between the two ring stages
for (current_ring in 1:nrow(apex_data_long)) {
  if (
    apex_data_long[current_ring,]$stage != 0 |
    apex_data_long[current_ring,]$stage != 5
  ) {
    angle_between[current_ring] <-
```

```
      atan(
        (
          apex_data_long[current_ring + 1,]$y -
            apex_data_long[current_ring,]$y
        ) /
          (
            apex_data_long[current_ring + 1,]$x -
              apex_data_long[current_ring,]$x
          )
      )
  }
}

# Transform any NaN into 0
angle_between[is.nan(angle_between)] <- 0

# Add the new information to the apex_data_long data set
apex_data_long <- apex_data_long %>%
  mutate(angle_between = angle_between)

# Save the long apex data to an RData file
save(apex_data_long, file = "rda/apex_data_long.rda")
```

2.3.2 Wide Data

Next, I want to create a wide representation of the Apex Legends match data, which I will use when training and evaluating the model for predicting the final ring location on a map. To do this, I will mutate the data set with similar information to the long representation of the data, but I will also make use of the `pivot_wider` function.

```
# Helper function to account for any NaN encountered within a data.frame
is.nan.data.frame <- function(df)
  do.call(cbind, lapply(df, is.nan))

# Copy the tidy apex data to a new variable for further manipulation
apex_data_wide <- apex_data_tidy
# Make a wide representation of the data set
# by making each row represent one whole game
# with all ring stage coordinates represented by their own columns
apex_data_wide <- apex_data_wide %>%
  filter(map == "World's Edge") %>%
  pivot_wider(
    names_from = stage,
    values_from = c(x, y, radius)
  ) %>%
  mutate(
    distance_from_center_1 =
      sqrt((x_1 - mean(apex_data_wide$x))^2 + (y_1 - mean(apex_data_wide$y))^2),
    distance_between_1_and_2 =
      sqrt((x_2 - x_1)^2 + (y_2 - y_1)^2),
    angle_between_1_and_2 =
      atan((y_2 - y_1) / (x_2 - x_1)),
    distance_from_center_2 =
      sqrt((x_2 - mean(apex_data_wide$x))^2 + (y_2 - mean(apex_data_wide$y))^2),
```

```
    distance_between_2_and_3 =
      sqrt((x_3 - x_2)^2 + (y_3 - y_2)^2),
    angle_between_2_and_3 =
      atan((y_3 - y_2) / (x_3 - x_2)),
    distance_from_center_3 =
      sqrt((x_3 - mean(apex_data_wide$x))^2 + (y_3 - mean(apex_data_wide$y))^2),
    distance_between_3_and_4 =
      sqrt((x_4 - x_3)^2 + (y_4 - y_3)^2),
    angle_between_3_and_4 =
      atan((y_4 - y_3) / (x_4 - x_3)),
    distance_from_center_4 =
      sqrt((x_4 - mean(apex_data_wide$x))^2 + (y_4 - mean(apex_data_wide$y))^2),
    distance_between_4_and_5 =
      sqrt((x_5 - x_4)^2 + (y_5 - y_4)^2),
    angle_between_4_and_5 =
      atan((y_5 - y_4) / (x_5 - x_4)),
    distance_from_center_5 =
      sqrt((x_5 - mean(apex_data_wide$x))^2 + (y_5 - mean(apex_data_wide$y))^2),
  )

apex_data_wide[is.nan(apex_data_wide)] <- 0

# Save the wide apex data to an RData file
save(apex_data_wide, file = "rda/apex_data_wide.rda")
```

Now each row in this data set represents one game as a whole, with columns corresponding to each x and y coordinate of each stage of the ring.

2.4 Visualizing Apex Legends Matches

Before I continue with the analysis portion of this capstone project, I would like to take a moment to look at the data I have transformed to see if I can notice any trends or tidbits of information that I might find useful later on when I am conducting machine learning.

```
# Summarize the radius data across rings on each map
apex_data_long %>%
  group_by(map, stage) %>% # Only compare rings within the same map
  summarize(
    radius_average = mean(radius),
    radius_standard_deviation = sd(radius),
    radius_min = min(radius),
    radius_max = max(radius)
  )
```

```
## # A tibble: 12 x 6
## # Groups:   map [2]
##    map         stage radius_average radius_standard_devi~1 radius_min radius_max
##    <fct>       <fct>          <dbl>                  <dbl>      <int>      <int>
##  1 Storm Point 0             12675.                  0.517      12672      12675
##  2 Storm Point 1              4492.                  0.495       4491       4493
##  3 Storm Point 2              2406.                  0.509       2405       2407
##  4 Storm Point 3              1283.                  0.462       1283       1284
##  5 Storm Point 4               642.                  0.271        641        642
##  6 Storm Point 5               321                   0             321        321
##  7 World's Ed~ 0             13283.                 67.2         12646      13291
```

```
##  8 World's Ed~ 1              4369.                  0.544       4368       4370
##  9 World's Ed~ 2              2367.                  0.479       2366       2367
## 10 World's Ed~ 3              1456.                  0.381       1456       1457
## 11 World's Ed~ 4               728                   0            728        728
## 12 World's Ed~ 5               364                   0            364        364
## # i abbreviated name: 1: radius_standard_deviation
```

```
# Summarize the ring area data across rings on each map
apex_data_long %>%
  group_by(map, stage) %>% # Only compare rings within the same map
  summarize(
    area_average = mean(pi * radius^2),
    area_standard_deviation = sd(pi * radius^2),
    area_min = min(pi * radius^2),
    area_max = max(pi * radius^2)
  )
```

```
## # A tibble: 12 x 6
## # Groups:   map [2]
##    map          stage area_average area_standard_deviation   area_min   area_max
##    <fct>        <fct>        <dbl>                   <dbl>      <dbl>      <dbl>
##  1 Storm Point  0       504696636.                  41157. 504475641. 504714531.
##  2 Storm Point  1        63393161.                  13982.  63363037.  63419485.
##  3 Storm Point  2        18191771.                   7690.  18171050.  18201285.
##  4 Storm Point  3         5173788.                   3728.   5171341.   5179406.
##  5 Storm Point  4         1294534.                   1091.   1290821.   1294851.
##  6 Storm Point  5          323713.                      0     323713.    323713.
##  7 World's Edge 0       554322415.                5472508. 502407631. 554964482.
##  8 World's Edge 1        59968122.                  14926.  59939778.  59994681.
##  9 World's Edge 2        17596195.                   7121.  17586497.  17601367.
## 10 World's Edge 3         6661567.                   3488.   6659975.   6669127.
## 11 World's Edge 4         1664994.                      0    1664994.   1664994.
## 12 World's Edge 5          416248.                      0     416248.    416248.
```

```
# Summarize the distance between rings data across rings on each map
apex_data_long %>%
  group_by(map, stage) %>% # Only compare rings within the same map
  summarize(
    distance_between_average = mean(distance_between),
    distance_between_standard_deviation = sd(distance_between),
    distance_between_min = min(distance_between),
    distance_between_max = max(distance_between)
  )
```

```
## # A tibble: 12 x 6
## # Groups:   map [2]
##    map          stage distance_between_average distance_between_standard_devia~1
##    <fct>        <fct>                    <dbl>                              <dbl>
##  1 Storm Point  0                       0.0608                              0.254
##  2 Storm Point  1                    2065.                                 79.2
##  3 Storm Point  2                    1061.                                183.
##  4 Storm Point  3                     474.                                210.
##  5 Storm Point  4                     260.                                 88.0
##  6 Storm Point  5                      NA                                  NA
##  7 World's Edge 0                       0.102                              0.378
##  8 World's Edge 1                    1964.                                109.
```

```
##  9 World's Edge 2                    781.                                  166.
## 10 World's Edge 3                    553.                                  185.
## 11 World's Edge 4                    239.                                  102.
## 12 World's Edge 5                   5192.                                 2173.
## # i abbreviated name: 1: distance_between_standard_deviation
## # i 2 more variables: distance_between_min <dbl>, distance_between_max <dbl>
```

```
# Summarize the angle between rings data across rings on each map
apex_data_long %>%
  group_by(map, stage) %>% # Only compare rings within the same map
  summarize(
    angle_between_average = mean(angle_between),
    angle_between_standard_deviation = sd(angle_between),
    angle_between_min = min(angle_between),
    angle_between_max = max(angle_between)
  )
```

```
## # A tibble: 12 x 6
## # Groups:   map [2]
##    map      stage angle_between_average angle_between_standa~1 angle_between_min
##    <fct>    <fct>                 <dbl>                  <dbl>             <dbl>
##  1 Storm P~ 0                  -0.0265                  0.301             -1.57
##  2 Storm P~ 1                   0.0770                  1.00              -1.56
##  3 Storm P~ 2                  -0.0492                  0.878             -1.52
##  4 Storm P~ 3                  -0.168                   0.931             -1.56
##  5 Storm P~ 4                   0.0694                  0.955             -1.55
##  6 Storm P~ 5                  NA                      NA                 NA
##  7 World's~ 0                   0.00854                 0.339             -1.57
##  8 World's~ 1                   0.0671                  0.949             -1.55
##  9 World's~ 2                   0.0805                  0.884             -1.55
## 10 World's~ 3                   0.00847                 0.891             -1.54
## 11 World's~ 4                  -0.174                   0.927             -1.55
## 12 World's~ 5                  -0.0144                  0.898             -1.56
## # i abbreviated name: 1: angle_between_standard_deviation
## # i 1 more variable: angle_between_max <dbl>
```

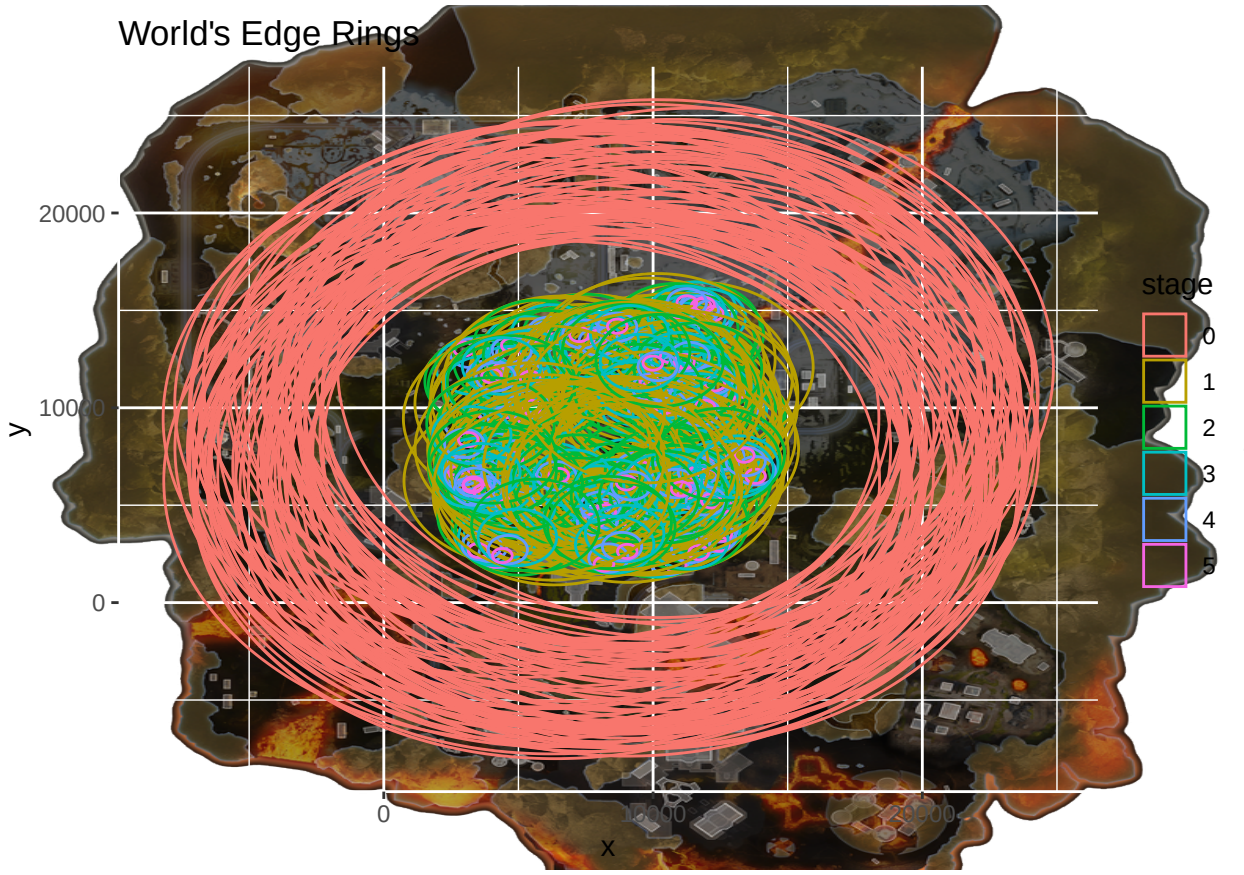
After summarizing the data, we can now see that:

1. The size of each stage of the ring are consistent across games on each map, meaning that the standard deviations of the radii of each ring on each map are very small
2. The distance between each stage of the ring is mostly consistent across games on each map, meaning that the standard deviations of the distances between each ring on each map is relatively small compared to the size of a map in Apex Legends
3. The angle between each stage of the ring is also mostly consistent across games on each map, meaning that the standard deviations of the angles between each ring on each map is relatively small compared to the size of a map in Apex Legends

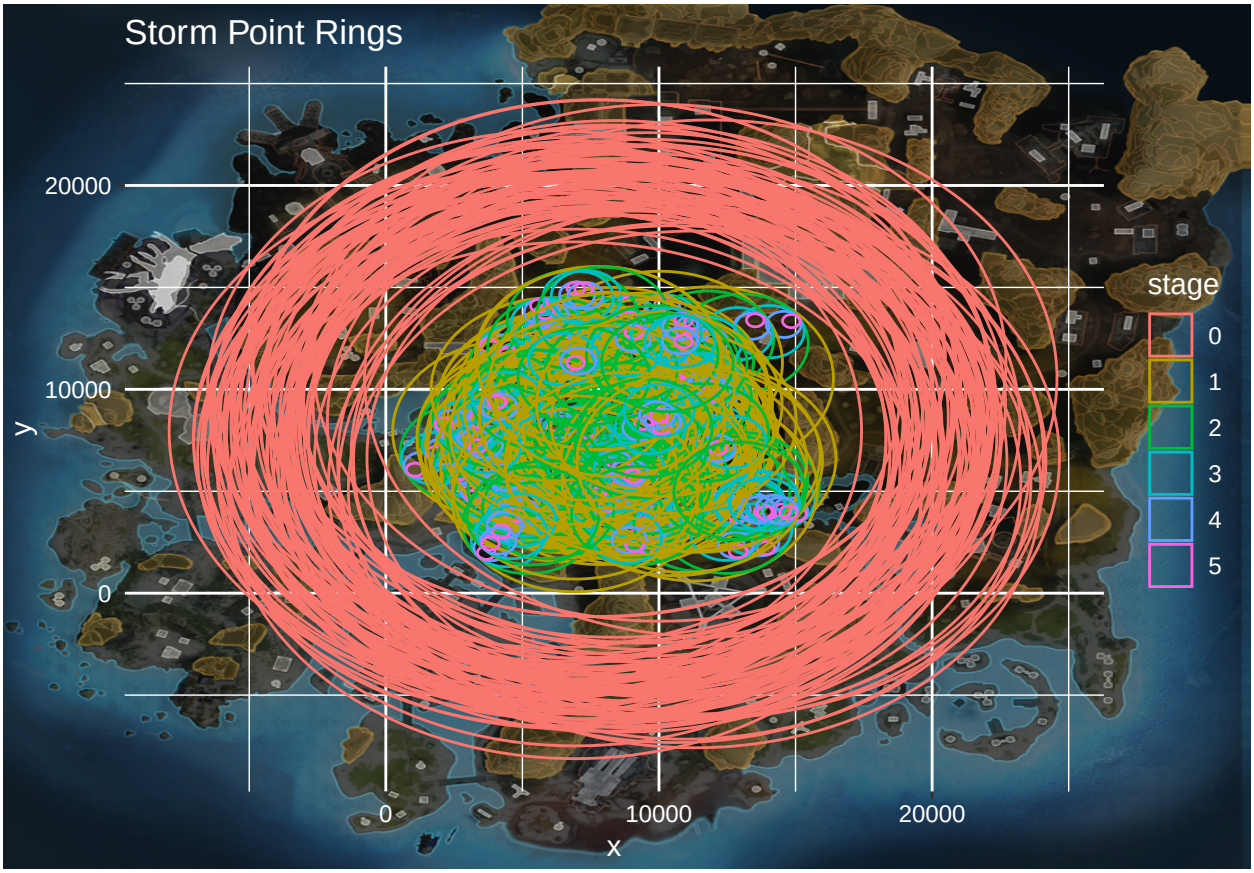
These three facts mean that I should be able to predict the final ring location in a game fairly accurately. Below are two plots superimposed on top of an image of each map to give the general idea of where each stage of the ring travels on each map.

```
# Visualize ring movement across all games on World's Edge
# Animated plot not shown in the report as gganimate doesn't work with knitr
worlds_edge_plot <- apex_data_long %>%
  filter(map == "World's Edge") %>%
  ggplot(aes(x0 = x, y0 = y, r = radius, group = gameID, color = stage)) +
  geom_circle() +
```

```
  ggtitle("World's Edge Rings") +
  theme(
    axis.line = element_line(color = "white")
  )
ggbackground(worlds_edge_plot, worlds_edge_image_file)
```




```
# Visualize ring movement across all games on Storm Point
# Animated plot not shown in the report as gganimate doesn't work with knitr
storm_point_plot <- apex_data_long %>%
  filter(map == "Storm Point") %>%
  ggplot(aes(x0 = x, y0 = y, r = radius, group = gameID, color = stage)) +
  geom_circle() +
  ggtitle("Storm Point Rings") +
  theme(
    axis.text = element_text(color = "white"),
    legend.text = element_text(color = "white"),
    title = element_text(color = "white")
  )
ggbackground(storm_point_plot, storm_point_image_file)
```



As you can see in the plots above, the ring tends to draw players towards the center of each map (which isn't always the case, but happens to be how each of these games turned out in the data set). The final ring demonstrates where highly contested POI are located on each map. Since it seems like I can anticipate the trajectory of each ring stage, this seems to be a good topic to train a machine learning model on to see how accurately I can predict the location of the final ring of an Apex Legends match.

2.5 Train and Test Sets

Since I am trying to predict two variables, the x and the y coordinate of the center of the final ring in an Apex Legends battle royale match, I will split the data into separate x and y train and test sets. I will also create two models and two sets of predictions later on.

```
# Create training and testing partitions of the wide apex data
# These sets are separate from the y train and test data sets
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
x_test_index <- createDataPartition(apex_data_wide$x_5, times = 1, p = 0.5, list = FALSE)
x_train_set <- apex_data_wide[-x_test_index,]
x_test_set <- apex_data_wide[x_test_index,]

# Create training and testing partitions of the wide apex data
# These sets are separate from the x train and test data sets
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
y_test_index <- createDataPartition(apex_data_wide$y_5, times = 1, p = 0.5, list = FALSE)
y_train_set <- apex_data_wide[-y_test_index,]
y_test_set <- apex_data_wide[y_test_index,]
```

After conducting several test runs with the `caret`(Kuhn and Max 2008) package, I decided on a 50/50 split between the train and test sets. This combination produced the best RMSE against the "Generalized Linear Model", which is the method I used for the baseline for the rest of my analysis. In this capstone, I will construct the following models and evaluate how well each mode predicts the location of the final ring in an Apex Legends match:

1. Generalized Linear Model
2. Generalized Linear Model with Stepwise Feature Selection
3. Least Angle Regression

3 Model Testing and Results

3.1 A Note On Accuracy

Because predicting the exact x and y coordinates out of these large maps will be incredibly difficult (if not impossible) for my models to do, I will evaluate the accuracy of my model's predictions on a sliding scale. I am interested to know, which predictions were within 250 meters? 100 meters? 50 meters? 25 meters? In the grand scheme of an Apex Legends match, these distances are relatively small, and this measure of accuracy will help determine whether to follow my model's predictions or not while playing a match of Apex Legends. Being in the general vicinity of the final ring long before the final ring closes in is an incredibly strategic move, and could make the difference between clinching the win or losing unprepared.

I will also use the Root Mean Squared Error (or RMSE for short) to help evaluate the fit of my model towards the problem at hand. RMSE is a loss function used to evaluate how well an algorithm predicts outcomes. Creating a model that minimizes the RMSE indicates that the model is making accurate predictions. Mathematically, RMSE is defined as:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

where $\hat{y}$ is a predicted outcome, y is an observed outcome, N is the number of observations made, and i is the current prediction/observation in the summation. We can calculate the RMSE with the following R code:

```
RMSE <- function(observed, prediction) {
  sqrt(mean((observed - prediction)^2))
}
```

The `caret`(Kuhn and Max 2008) package also contains a function to calculate the RMSE:

```
?caret::RMSE()
```

In this case, however, since I am already using the `caret`(Kuhn and Max 2008) package to create the models, the model object already contains the RMSE information, which means there is no need to calculate it separately.

3.2 Generalized Linear Model

The first method I am going to use to train my models to predict the location of the final ring in an Apex Legends match will be the "Generalized Linear Model" method, or `glm` for short. `glm` is essentially linear regression with the main difference being that the "Generalized Linear Model" can fit more complex models than just regular linear regression. To see more information about this method, you can execute the R code seen below.

```
getModelInfo("glm")
```

This method has no tuning parameters to tweak how the algorithm is run in the back-end. However, this should give me a good baseline with which I can continue my investigation. In the following R code, I will

train two models separately, one for predicting x coordinates and another for predicting y coordinates. I will then use these two separate models to make distinct predictions for x and y coordinates. After these models make their predictions, I will evaluate their predictions' accuracy based on the sliding scale discussed above against the actual x and y coordinates of the final ring location in the corresponding games.

```
# Generalized Linear Model
# Train a model to predict the x coordinate
# of the final stage of the ring in an Apex Legends match
x_model <- train(
  x_5 ~ x_0 + x_1 + x_2 + x_3 + x_4,
  data = x_train_set,
  method = "glm"
)

# Make predictions for x_5 in the x_test_set based on the x_model
x_predictions <- predict(x_model, newdata = x_test_set)

# Since I'm not looking to be exact,
# evaluate the accuracy of the predictions on a sliding scale
glm_x_accuracy_within_250_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 250 &
      x_predictions >= x_test_set$x_5 - 250
  )
glm_x_accuracy_within_100_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 100 &
      x_predictions >= x_test_set$x_5 - 100
  )
glm_x_accuracy_within_50_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 50 &
      x_predictions >= x_test_set$x_5 - 50
  )
glm_x_accuracy_within_25_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 25 &
      x_predictions >= x_test_set$x_5 - 25
  )

# Train a model to predict the y coordinate
# of the final stage of the ring in an Apex Legends match
y_model <- train(
  y_5 ~ y_0 + y_1 + y_2 + y_3 + y_4,
  data = y_train_set,
  method = "glm"
)

# Make predictions for y_5 in the y_test_set based on the y_model
y_predictions <- predict(y_model, newdata = y_test_set)

# Since I'm not looking to be exact, evaluate the accuracy of the predictions on a sliding scale
glm_y_accuracy_within_250_meters <-
  mean(
```

```
    y_predictions <= y_test_set$y_5 + 250 &
      y_predictions >= y_test_set$y_5 - 250
  )
glm_y_accuracy_within_100_meters <-
  mean(
    y_predictions <= y_test_set$y_5 + 100 &
      y_predictions >= y_test_set$y_5 - 100
  )
glm_y_accuracy_within_50_meters <-
  mean(
    y_predictions <= y_test_set$y_5 + 50 &
      y_predictions >= y_test_set$y_5 - 50
  )
glm_y_accuracy_within_25_meters <-
  mean(
    y_predictions <= y_test_set$y_5 + 25 &
      y_predictions >= y_test_set$y_5 - 25
  )

# Print and save the accuracy results to a results variable
accuracy_results <- data.frame(
  method = "glm",
  x_250 = glm_x_accuracy_within_250_meters,
  x_100 = glm_x_accuracy_within_100_meters,
  x_50 = glm_x_accuracy_within_50_meters,
  x_25 = glm_x_accuracy_within_25_meters,
  y_250 = glm_y_accuracy_within_250_meters,
  y_100 = glm_y_accuracy_within_100_meters,
  y_50 = glm_y_accuracy_within_50_meters,
  y_25 = glm_y_accuracy_within_25_meters
)
accuracy_results %>% knitr::kable()
```

method	x_250	x_100	x_50	x_25	y_250	y_100	y_50	y_25
glm	0.875	0.3958333	0.2083333	0.1666667	0.7708333	0.2916667	0.0833333	0.0416667

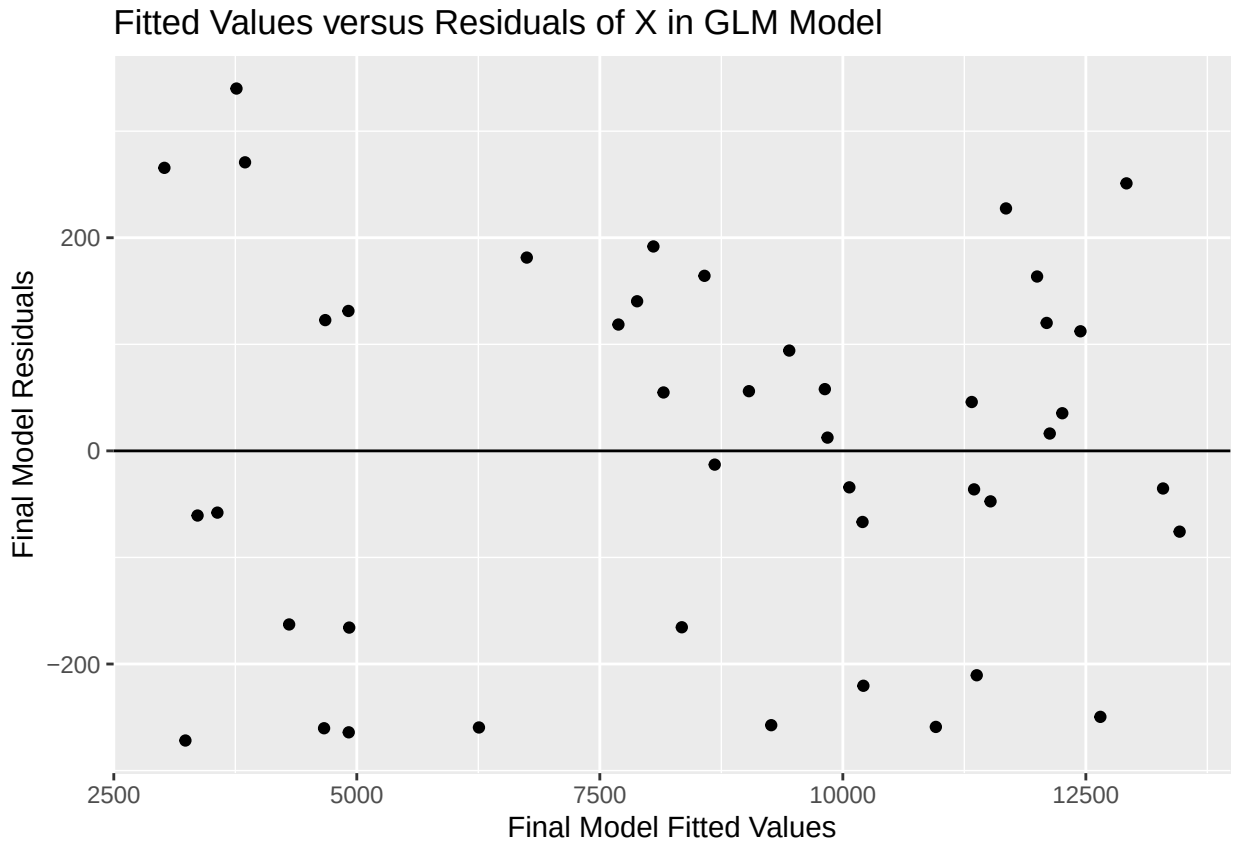
```
# Save the results to a RData file
save(accuracy_results, file = "rda/accuracy_results.rda")

rmse_results <- data.frame(
  method = "glm",
  x_RMSE = x_model$results$RMSE,
  y_RMSE = y_model$results$RMSE
)
rmse_results %>% knitr::kable()
```

method	x_RMSE	y_RMSE
glm	222.6889	230.7225

```r
# Save the results to a RData file
save(rmse_results, file = "rda/rmse_results.rda")
```

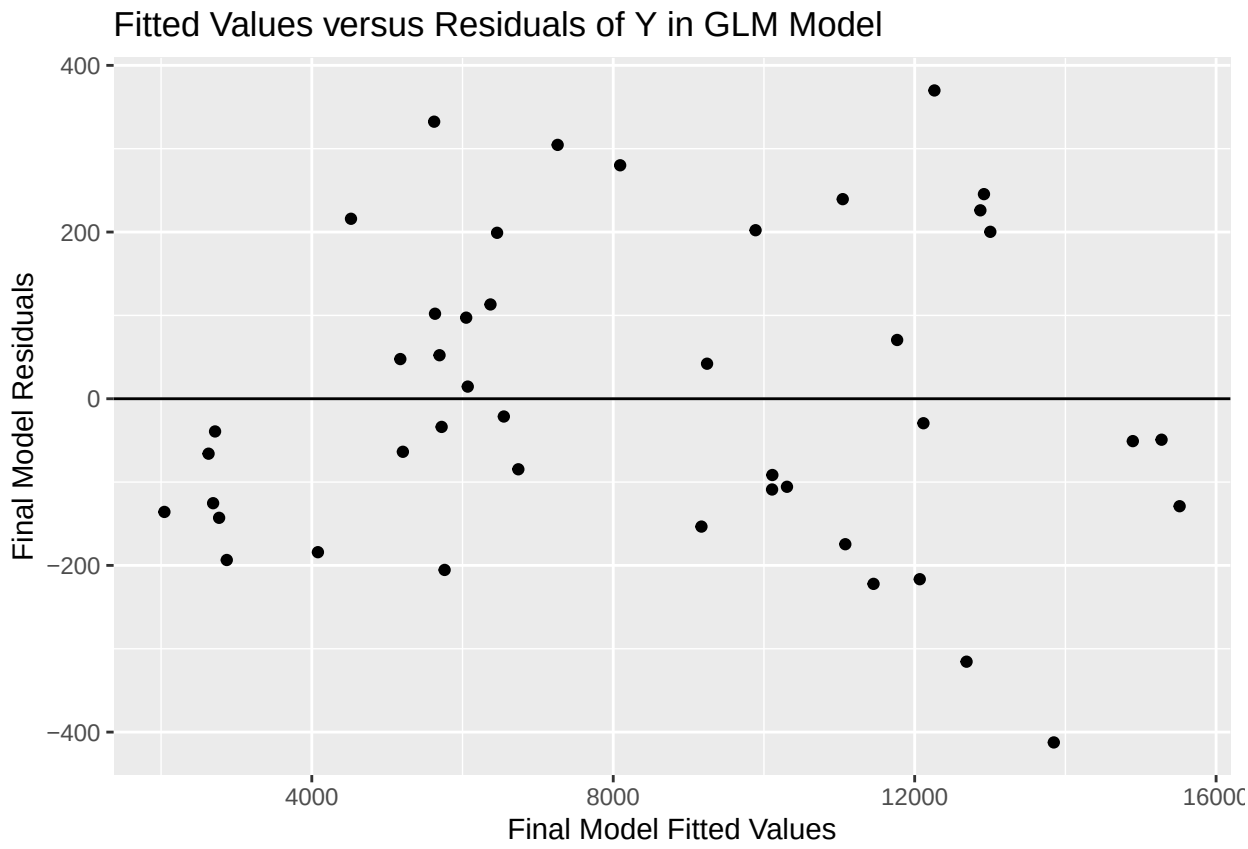
As we can see, these basic models have an accuracy in the upper 80s for within 250 meters, give or take, which I would say is very good. Movement characters (like Valkyrie) can easily reposition your squad hundreds of meters at a time, so these predictions will get us pretty much right where we need to be. The RMSEs is not close to zero, but because we're not looking for pinpoint accuracy in this project, the RMSEs here are still in a good spot.

```r
# Plot the fitted against the residuals of the x_model
# to see if there are any trends in the variance
# This is a test to see if there are any problems with the x_model
x_model$finalModel %>%
  ggplot(aes(.fitted, .resid)) +
  geom_point() +
  geom_hline(yintercept = 0) +
  xlab("Final Model Fitted Values") +
  ylab("Final Model Residuals") +
  ggtitle("Fitted Values versus Residuals of X in GLM Model")
```




```r
# Plot the fitted against the residuals of the y_model
# to see if there are any trends in the variance
# This is a test to see if there are any problems with the y_model
y_model$finalModel %>%
  ggplot(aes(.fitted, .resid)) +
  geom_point() +
  geom_hline(yintercept = 0) +
```

```
  xlab("Final Model Fitted Values") +
  ylab("Final Model Residuals") +
  ggtitle("Fitted Values versus Residuals of Y in GLM Model")
```



The plots above demonstrate that the variance across the `x_model` and `y_model` are consistent, meaning that the model is not heavily favoring high values or low values and that there are no patterns across the random observed variables. This means that there are no evident problems with this "Generalized Linear Model".

3.3 Generalized Linear Model With Stepwise Feature Selection

Taking the "Generalized Linear Model" to the next level, here I decided to try out the "Generalized Linear Model With Stepwise Feature Selection". You can view more information about this method by executing the R code shown below.

```
getModelInfo("glmStepAIC")
```

This method is similar to the "Generalized Linear Model" method, except that it automatically chooses the optimal set of predictors through an iterative testing process. With the verbose output enabled, you can see the different combinations of supplied predictors this algorithm creates and tries out.

```
# Generalized Linear Model With Stepwise Feature Selection
# Train a model to predict the x coordinate
# of the final stage of the ring in an Apex Legends match
x_model <- train(
  x_5 ~ x_0 + x_1 + x_2 + x_3 + x_4,
  data = x_train_set,
  method = "glmStepAIC",
```

```
  trace = 0 # Suppress verbose output
)

# Make predictions for x_5 in the x_test_set based on the x_model
x_predictions <- predict(x_model, newdata = x_test_set)

# Since I'm not looking to be exact,
# evaluate the accuracy of the predictions on a sliding scale
glmStepAIC_x_accuracy_within_250_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 250 &
      x_predictions >= x_test_set$x_5 - 250
  )
glmStepAIC_x_accuracy_within_100_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 100 &
      x_predictions >= x_test_set$x_5 - 100
  )
glmStepAIC_x_accuracy_within_50_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 50 &
      x_predictions >= x_test_set$x_5 - 50
  )
glmStepAIC_x_accuracy_within_25_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 25 &
      x_predictions >= x_test_set$x_5 - 25
  )

# Train a model to predict the y coordinate
# of the final stage of the ring in an Apex Legends match
y_model <- train(
  y_5 ~ y_0 + y_1 + y_2 + y_3 + y_4,
  data = y_train_set,
  method = "glmStepAIC",
  trace = 0 # Suppress verbose output
)

# Make predictions for y_5 in the y_test_set based on the y_model
y_predictions <- predict(y_model, newdata = y_test_set)

# Since I'm not looking to be exact,
# evaluate the accuracy of the predictions on a sliding scale
glmStepAIC_y_accuracy_within_250_meters <-
  mean(
    y_predictions <= y_test_set$y_5 + 250 &
      y_predictions >= y_test_set$y_5 - 250
  )
glmStepAIC_y_accuracy_within_100_meters <-
  mean(
    y_predictions <= y_test_set$y_5 + 100 &
      y_predictions >= y_test_set$y_5 - 100
  )
```

```r
glmStepAIC_y_accuracy_within_50_meters <-
  mean(
    y_predictions <= y_test_set$y_5 + 50 &
      y_predictions >= y_test_set$y_5 - 50
  )
glmStepAIC_y_accuracy_within_25_meters <-
  mean(
    y_predictions <= y_test_set$y_5 + 25 &
      y_predictions >= y_test_set$y_5 - 25
  )

# Print and save the accuracy results to a results variable
accuracy_results <- bind_rows(
  accuracy_results,
  data.frame(
    method = "glmStepAIC",
    x_250 = glmStepAIC_x_accuracy_within_250_meters,
    x_100 = glmStepAIC_x_accuracy_within_100_meters,
    x_50 = glmStepAIC_x_accuracy_within_50_meters,
    x_25 = glmStepAIC_x_accuracy_within_25_meters,
    y_250 = glmStepAIC_y_accuracy_within_250_meters,
    y_100 = glmStepAIC_y_accuracy_within_100_meters,
    y_50 = glmStepAIC_y_accuracy_within_50_meters,
    y_25 = glmStepAIC_y_accuracy_within_25_meters
  )
)
accuracy_results %>% knitr::kable()
```

method	x_250	x_100	x_50	x_25	y_250	y_100	y_50	y_25
glm	0.8750000	0.3958333	0.2083333	0.1666667	0.7708333	0.2916667	0.0833333	0.0416667
glmStepAIC	0.8541667	0.4375000	0.2708333	0.1666667	0.7708333	0.2916667	0.1458333	0.0833333

```r
# Save the results to a RData file
save(accuracy_results, file = "rda/accuracy_results.rda")

rmse_results <- bind_rows(
  rmse_results,
  data.frame(
    method = "glmStepAIC",
    x_RMSE = x_model$results$RMSE,
    y_RMSE = y_model$results$RMSE
  )
)
rmse_results %>% knitr::kable()
```

method	x_RMSE	y_RMSE
glm	222.6889	230.7225
glmStepAIC	192.4899	226.0049

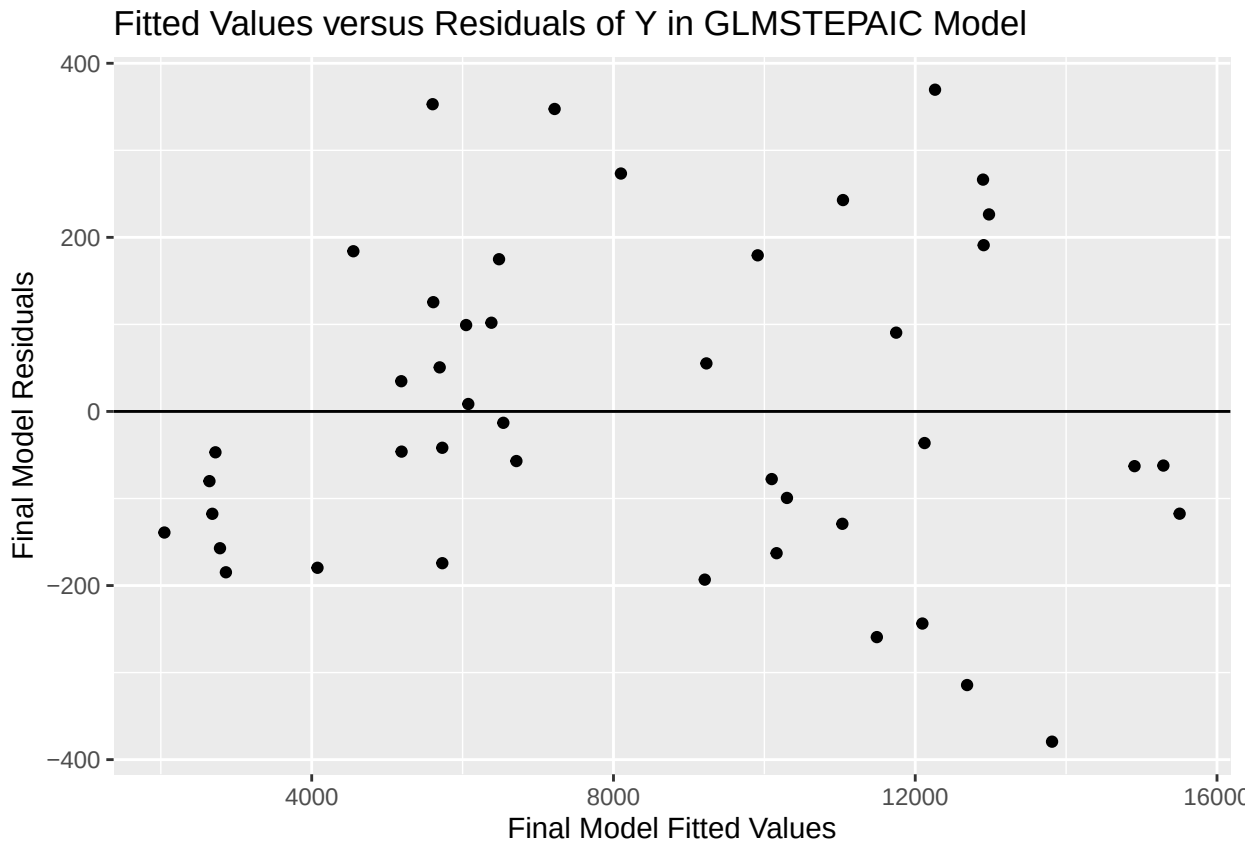
```
# Save the results to a RData file
save(rmse_results, file = "rda/rmse_results.rda")
```

The rest of the R code is similar to the "Generalized Linear Model" code, where I gather the accuracy and and RMSEs of each model's predictions. Below is the plot displaying the predicted final ring locations against the actual final ring locations.

```
# Plot the fitted against the residuals of the x_model
# to see if there are any trends in the variance
# This is a test to see if there are any problems with the x_model
x_model$finalModel %>%
  ggplot(aes(.fitted, .resid)) +
  geom_point() +
  geom_hline(yintercept = 0) +
  xlab("Final Model Fitted Values") +
  ylab("Final Model Residuals") +
  ggtitle("Fitted Values versus Residuals of X in GLMSTEPAIC Model")
```


```
# Plot the fitted against the residuals of the y_model
# to see if there are any trends in the variance
# This is a test to see if there are any problems with the y_model
y_model$finalModel %>%
  ggplot(aes(.fitted, .resid)) +
  geom_point() +
  geom_hline(yintercept = 0) +
  xlab("Final Model Fitted Values") +
  ylab("Final Model Residuals") +
```

```
ggtitle("Fitted Values versus Residuals of Y in GLMSTEPAIC Model")
```



Again, here are the plots that test the "Generalized Linear Model with Stepwise Feature Selection" models for any issues in variance, which I do not detect here.

3.4 Cubist Regression

After testing a large number of algorithms made available through the `caret`(Kuhn and Max 2008), I decided on using the "Cubist Regression" method, or `cubist` for short. You can see more information about this method as well by executing the R code shown below.

```
getModelInfo("cubist")
```

This is a different type of regression from the linear regression we are used to seeing. Here, the "Cubist Regression" will a tree where each leaf on the tree is a linear regression. The "Cubist Regression" smooths the predictions taking into account previous linear regressions recursively all the way back up to the root of the tree. In the end, this tree creates a set of rules with which the model will make its predictions. I chose this algorithm out of all the others that I tested because:

1. The RMSE most closely resembled those that have already been shown in this capstone, and
2. Since I have proven thus far that it is possible to , reasonably, there must be a set of rules that Apex Legends uses to decide where the location of the final ring will fall. This is especially true in the Apex Legends Global Series (ALGS), the global tournament held throughout the year for Apex Legends battle royale where e-sports teams from all over the Earth battle it out to be the best of the best at Apex Legends. The ring destinations in each match for the ALGS are predetermined to create the most interesting situations in each game, which means there are rules that ultimately decide the location of the final ring in an Apex Legends match.

```r
# Cubist Regression
# Train a model to predict the x coordinate
# of the final stage of the ring in an Apex Legends match
x_model <- train(
  x_5 ~ x_0 + x_1 + x_2 + x_3 + x_4,
  data = x_train_set,
  method = "cubist"
)

# Make predictions for x_5 in the x_test_set based on the x_model
x_predictions <- predict(x_model, newdata = x_test_set)

# Since I'm not looking to be exact,
# evaluate the accuracy of the predictions on a sliding scale
cubist_x_accuracy_within_250_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 250 &
      x_predictions >= x_test_set$x_5 - 250
  )
cubist_x_accuracy_within_100_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 100 &
      x_predictions >= x_test_set$x_5 - 100
  )
cubist_x_accuracy_within_50_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 50 &
      x_predictions >= x_test_set$x_5 - 50
  )
cubist_x_accuracy_within_25_meters <-
  mean(
    x_predictions <= x_test_set$x_5 + 25 &
      x_predictions >= x_test_set$x_5 - 25
  )

# Train a model to predict the y coordinate
# of the final stage of the ring in an Apex Legends match
y_model <- train(
  y_5 ~ y_0 + y_1 + y_2 + y_3 + y_4,
  data = y_train_set,
  method = "cubist"
)

# Make predictions for y_5 in the y_test_set based on the y_model
y_predictions <- predict(y_model, newdata = y_test_set)

# Since I'm not looking to be exact,
# evaluate the accuracy of the predictions on a sliding scale
cubist_y_accuracy_within_250_meters <-
  mean(
    y_predictions <= y_test_set$y_5 + 250 &
      y_predictions >= y_test_set$y_5 - 250
  )
```

```
cubist_y_accuracy_within_100_meters <-
  mean(
    y_predictions <= y_test_set$y_5 + 100 &
      y_predictions >= y_test_set$y_5 - 100
  )
cubist_y_accuracy_within_50_meters <-
  mean(
    y_predictions <= y_test_set$y_5 + 50 &
      y_predictions >= y_test_set$y_5 - 50
  )
cubist_y_accuracy_within_25_meters <-
  mean(
    y_predictions <= y_test_set$y_5 + 25 &
      y_predictions >= y_test_set$y_5 - 25
  )

# Print and save the accuracy results to a results variable
accuracy_results <- bind_rows(
  accuracy_results,
  data.frame(
    method = "cubist",
    x_250 = cubist_x_accuracy_within_250_meters,
    x_100 = cubist_x_accuracy_within_100_meters,
    x_50 = cubist_x_accuracy_within_50_meters,
    x_25 = cubist_x_accuracy_within_25_meters,
    y_250 = cubist_y_accuracy_within_250_meters,
    y_100 = cubist_y_accuracy_within_100_meters,
    y_50 = cubist_y_accuracy_within_50_meters,
    y_25 = cubist_y_accuracy_within_25_meters
  )
)
accuracy_results %>% knitr::kable()
```

method	x_250	x_100	x_50	x_25	y_250	y_100	y_50	y_25
glm	0.8750000	0.3958333	0.2083333	0.1666667	0.7708333	0.2916667	0.0833333	0.0416667
glmStepAIC	0.8541667	0.4375000	0.2708333	0.1666667	0.7708333	0.2916667	0.1458333	0.0833333
cubist	0.8333333	0.4375000	0.3125000	0.1875000	0.7916667	0.3541667	0.1666667	0.0833333

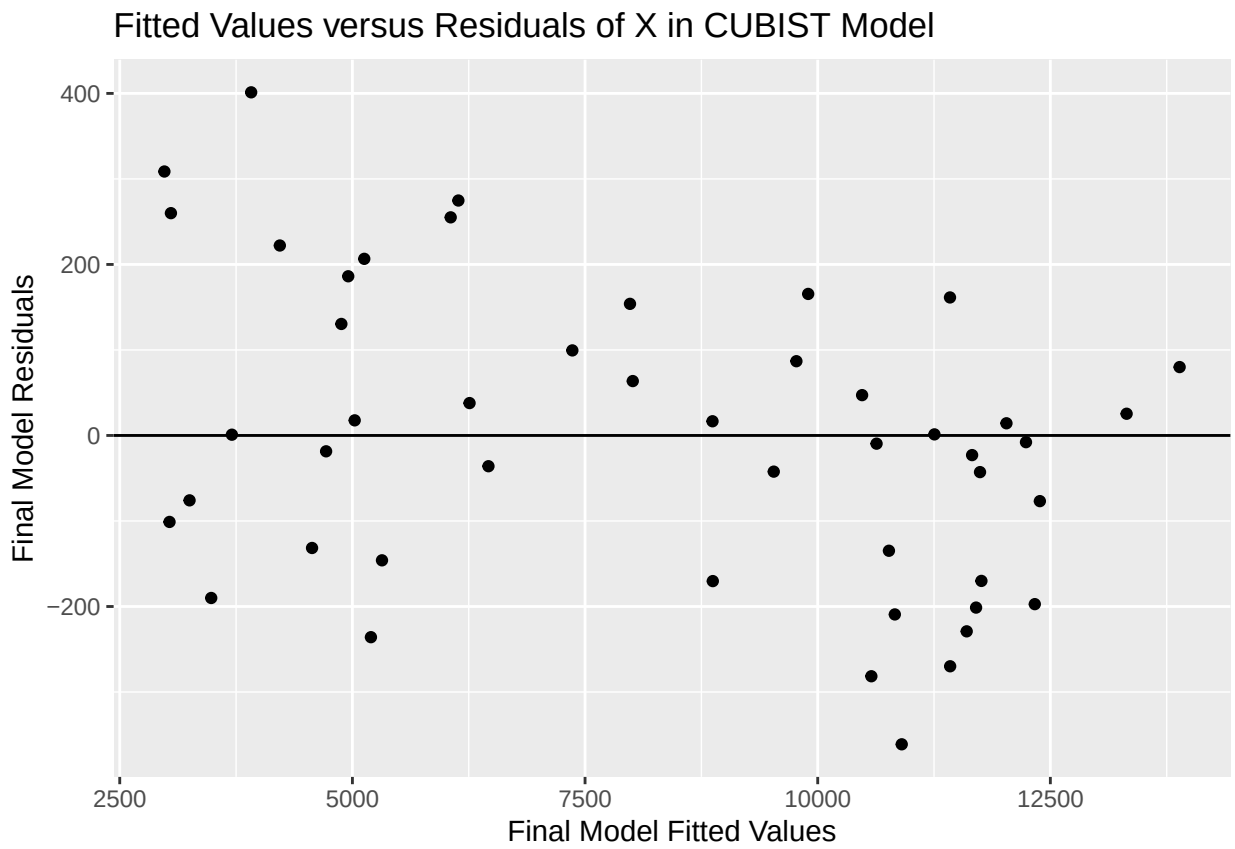
```
# Save the results to a RData file
save(accuracy_results, file = "rda/accuracy_results.rda")

rmse_results <- bind_rows(
  rmse_results,
  data.frame(
    method = "cubist",
    x_RMSE = x_model$results$RMSE,
    y_RMSE = y_model$results$RMSE
  )
)
rmse_results %>% knitr::kable()
```

method	x_RMSE	y_RMSE
glm	222.6889	230.7225
glmStepAIC	192.4899	226.0049
cubist	234.1956	242.4424
cubist	235.5671	237.6500
cubist	231.1780	236.2700
cubist	218.8688	216.7926
cubist	223.5754	216.8475
cubist	218.4852	215.9260
cubist	225.2038	219.4500
cubist	228.7851	218.6328
cubist	223.8147	217.9230

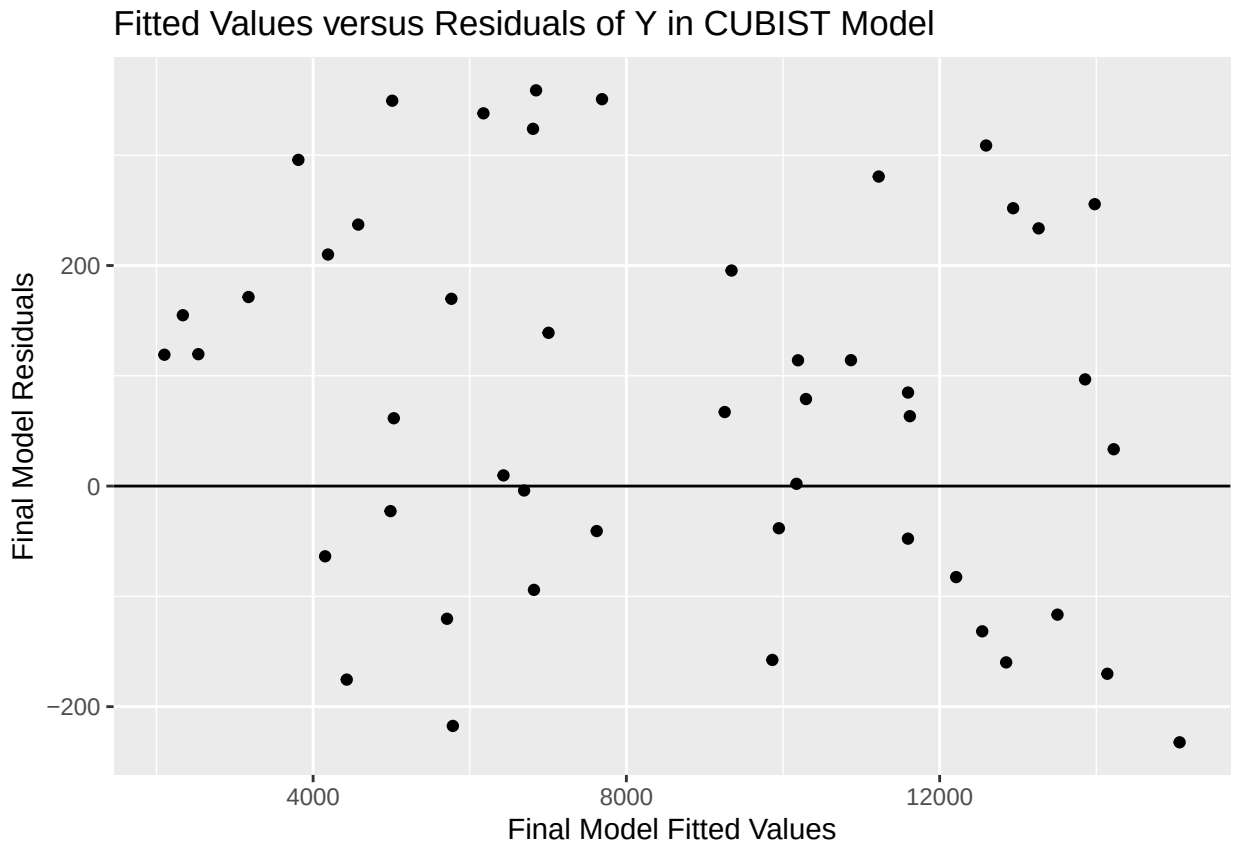
```
# Save the results to a RData file
save(rmse_results, file = "rda/rmse_results.rda")
```

Again, the rest of the R code should be familiar, where I gather the accuracy and and RMSEs of each model's predictions. Also, below is the usual plot displaying if there are any patterns to the variance in the model, of which there are none.

```
# Plot the fitted against the residuals of the x_model
# to see if there are any trends in the variance
# This is a test to see if there are any problems with the x_model
# Since "cubist" comes from a different library
# I have to create my own data frame for plotting
data.frame(
  fitted = x_predictions,
  resid = x_predictions - x_test_set$x_5
) %>%
  ggplot(aes(fitted, resid)) +
  geom_point() +
  geom_hline(yintercept = 0) +
  xlab("Final Model Fitted Values") +
  ylab("Final Model Residuals") +
  ggtitle("Fitted Values versus Residuals of X in CUBIST Model")
```




```
# Plot the fitted against the residuals of the y_model
# to see if there are any trends in the variance
# This is a test to see if there are any problems with the y_model
# Since "cubist" comes from a different library
# I have to create my own data frame for plotting
data.frame(
  fitted = y_predictions,
  resid = y_predictions - y_test_set$y_5
) %>%
  ggplot(aes(fitted, resid)) +
  geom_point() +
  geom_hline(yintercept = 0) +
  xlab("Final Model Fitted Values") +
  ylab("Final Model Residuals") +
  ggtitle("Fitted Values versus Residuals of Y in CUBIST Model")
```



4 Conclusion

After all the iterations on the methods used to train the algorithm, here are where the accuracy of each model stands.

method	x_250	x_100	x_50	x_25	y_250	y_100	y_50	y_25
glm	0.8750000	0.3958333	0.2083333	0.1666667	0.7708333	0.2916667	0.0833333	0.0416667
glmStepAIC	0.8541667	0.4375000	0.2708333	0.1666667	0.7708333	0.2916667	0.1458333	0.0833333
cubist	0.8333333	0.4375000	0.3125000	0.1875000	0.7916667	0.3541667	0.1666667	0.0833333

Also, here are where the RMSE results for each method stand.

method	x_RMSE	y_RMSE
glm	222.6889	230.7225
glmStepAIC	192.4899	226.0049
cubist	234.1956	242.4424
cubist	235.5671	237.6500
cubist	231.1780	236.2700
cubist	218.8688	216.7926
cubist	223.5754	216.8475
cubist	218.4852	215.9260
cubist	225.2038	219.4500
cubist	228.7851	218.6328

method	x_RMSE	y_RMSE
cubist	223.8147	217.9230

As we can see, the "Generalized Linear Model With Stepwise Feature Selection" had the best RMSE for the x coordinate, an iteration of the "Cubist Regression" had the best RMSE for the y coordinate, and both the "Generalized Linear Model With Stepwise Feature Selection" and "Least Angle Regression" had similar scores when it came to accuracy across the board. This indicates to me that, in the future, an ensemble algorithm, where multiple algorithms work together, would best suit a model for predicting the final ring position in an Apex Legends match, and that the same algorithm is not necessarily the best fit for the x and the y coordinates of said ring position.

4.1 Future Considerations

In the future, I would be interested to see what other features I can add to my data sets and see if they affect my model at all. I would also love to conduct this research on the other battle royale maps in Apex Legends. Additionally, it would be incredible to put my model to the test in a live Apex Legends battle royale match. Lastly, I plan on sharing this experiment with my brother to see what he has to say, since he's much better at the game and been playing Apex Legends for much longer than I have.

4.2 Final Remarks

Thanks to all who read through this report and look through my repository. I look forward to applying my newly gained machine learning knowledge on future projects, and hope to continue expanding that knowledge in this ever-growing field.

References

"Apex Legends - About." 2019. https://www.ea.com/games/apex-legends/about.

"Apex Legends - Battle Royale." 2019. https://www.ea.com/games/apex-legends/modes/battle-royale.

"Apex Legends - Battle Royale - Maps." 2019. https://www.ea.com/games/apex-legends/maps#battle-royale.

"Apex Legends - Characters." 2019. https://www.ea.com/games/apex-legends/about/characters.

"Apex Legends - Endzone & Ring Prediction." 2022. https://github.com/ccamfpsApex/ApexLegendsGuide/wiki/Endzone-&-Ring-Prediction.

"Apex Zone Predict Machine Learning." 2023. https://github.com/bluelightgit/apex-zone-predict-machine-learning/tree/main.

Auguie, Baptiste. 2017. *gridExtra: Miscellaneous Functions for "Grid" Graphics*. https://CRAN.R-project.org/package=gridExtra.

Kuhn, and Max. 2008. "Building Predictive Models in r Using the Caret Package." *Journal of Statistical Software* 28 (5): 1–26. https://doi.org/10.18637/jss.v028.i05.

Ooms, Jeroen. 2014. "The Jsonlite Package: A Practical and Consistent Mapping Between JSON Data and r Objects." *arXiv:1403.2805 [Stat.CO]*. https://arxiv.org/abs/1403.2805.

Pedersen, Thomas Lin. 2022. *Ggforce: Accelerating 'Ggplot2'*. https://CRAN.R-project.org/package=ggforce.

Pedersen, Thomas Lin, and David Robinson. 2022. *Gganimate: A Grammar of Animated Graphics*. https://CRAN.R-project.org/package=gganimate.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemun, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Yu, Guangchuang. 2023. *Ggimage: Use Image in 'Ggplot2'*. https://CRAN.R-project.org/package=ggimage.